Model-based Network Fault Injection for IoT Protocols

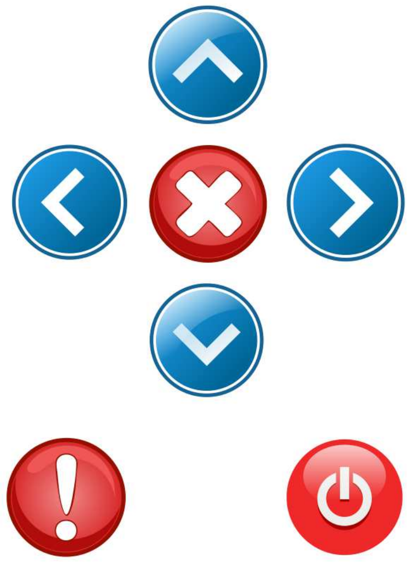
Jun Yoneyama
The University of Tokyo, Japan
Cyrille Artho
KTH Royal Institute of Technology, Stockholm, Sweden
`artho@kth.se`
Yoshinori Tanabe, Masami Hagiya
Tsurumi University and The University of Tokyo, Japan

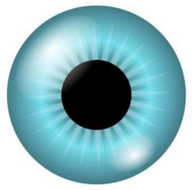
Software Testing

Input

System

Output/
observation





Can this be automated?

Unit Testing

```
@Test void test1() {
    pos = p0;
    left();
    right();
    assert(pos == p0);
}
```

```
@Test void test2() {
    pos = p0;
    right();
    left();
    assert(pos == p0);
}
```

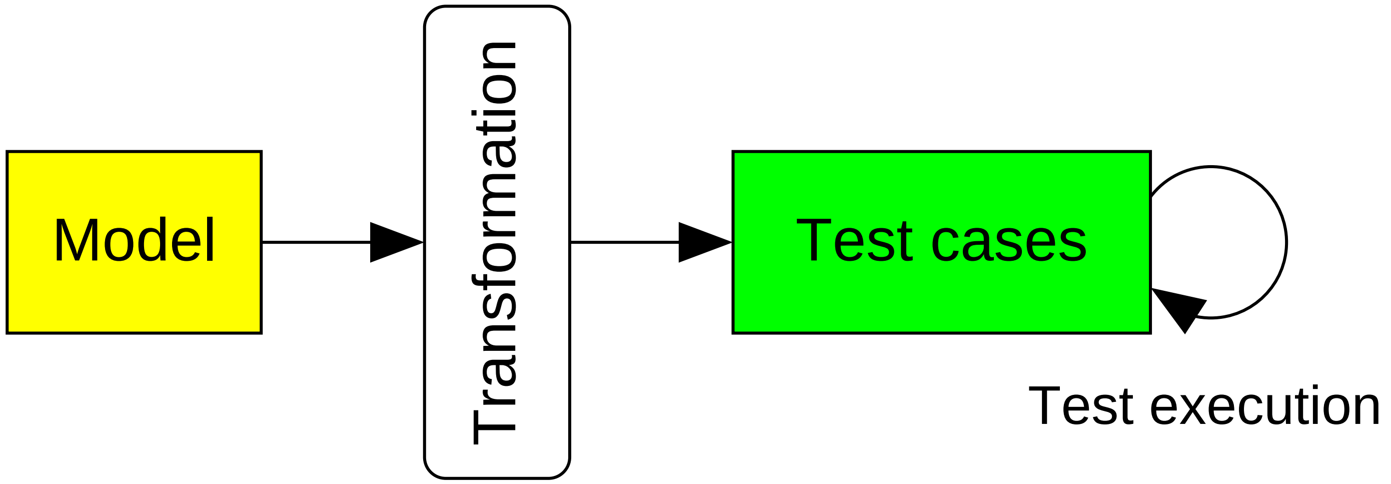
```
@Test void test3() {
    pos = p0;
    left();
    left();
    right();
    right();
    assert(pos == p0);
}
```

```
@Test void test4() {
    pos = p0;
    left();
    right();
    right();
    left();
    assert(pos == p0);
}
```

Can this be automated?

Overview

Model-based Testing

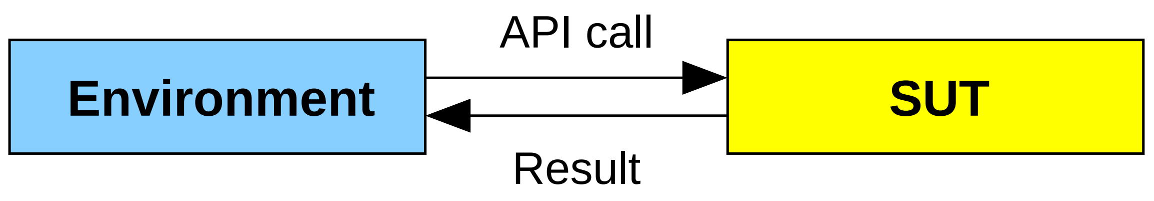


- Model contains:

  → Formalized description of the system behavior.
  → Input, expected output, exceptions, state.

- Transformation tool generates and executes test cases (on-line).

Test Model vs. System Model

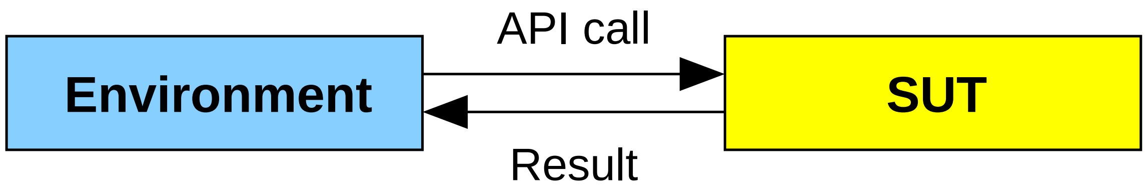


SUT = System under test; API = Application programming interface

Test model **System model**

What **How**

Test Model vs. System Model



SUT = System under test; API = Application programming interface

Test model

- Represents **environment**.
- Models system **behavior**.
- Used to generate **test** cases.
- Model, test one module at a time;
  SUT itself provides counterpart.
- Model-based testing.

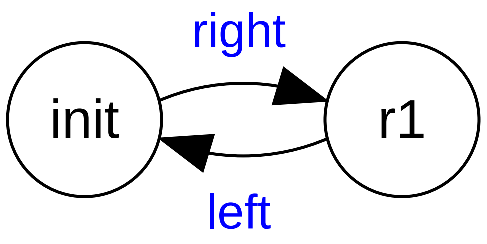
System model

- Represents **system** itself.
- Models system **implementation**.
- Used to **verify** system.
- Need model of most components
  to analyze system behavior.
- Model checking, theorem proving.

Modeling tests with Modbat

Domain-Specific Language (DSL) based on Scala.

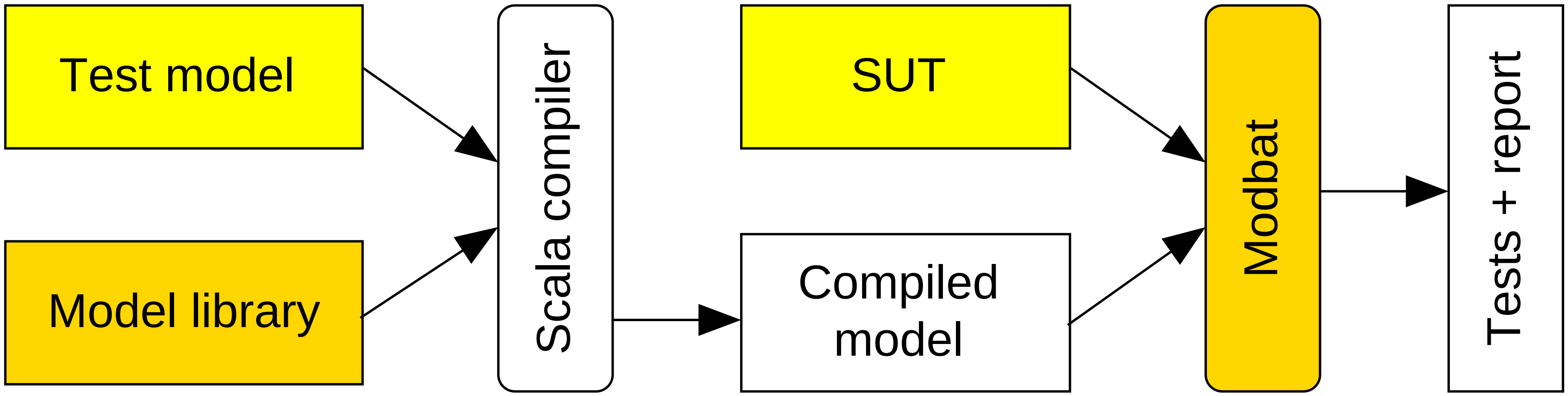
- Extended Finite-State Machine (EFSM) as base structure.
- Add transition functions, variables for complex state.
- Structured model but flexibility of full Scala (+ Java).



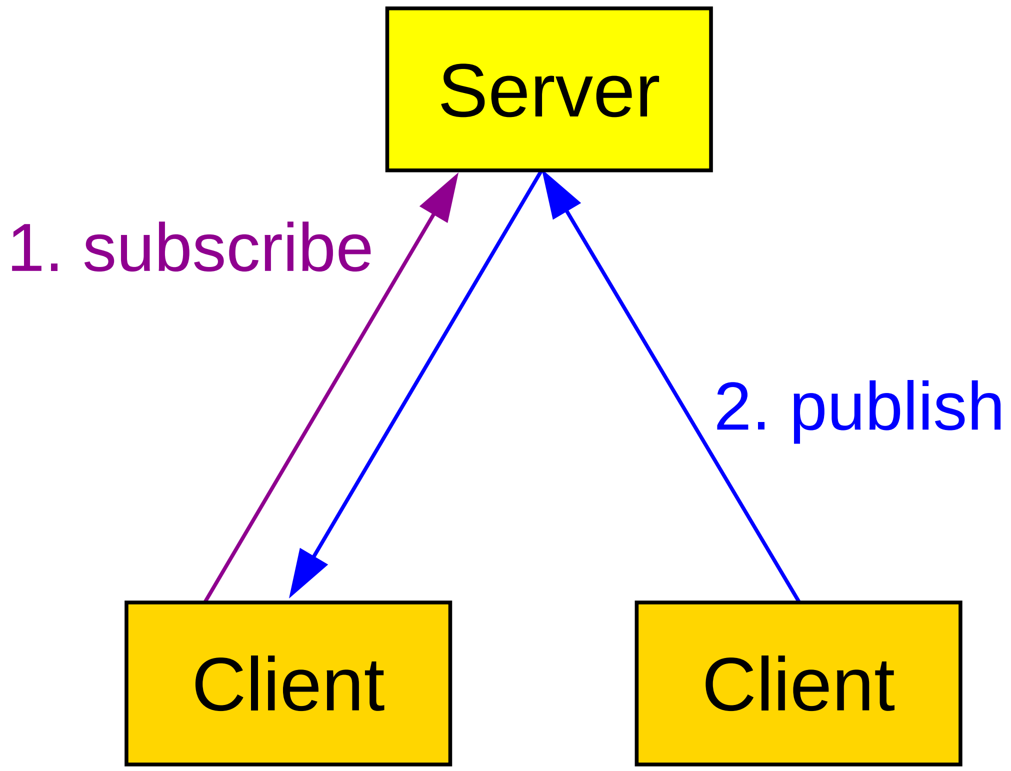
```
class Example extends Model {
  var r = 0
  "init" -> "r1" := { right; r += 1 }
  "r1" -> "init" := { left; assert (r > 0) }
}
```

Architecture and Workflow of Modbat

1. User defines test model.
2. Modbat executes tests from model against system under test (SUT).

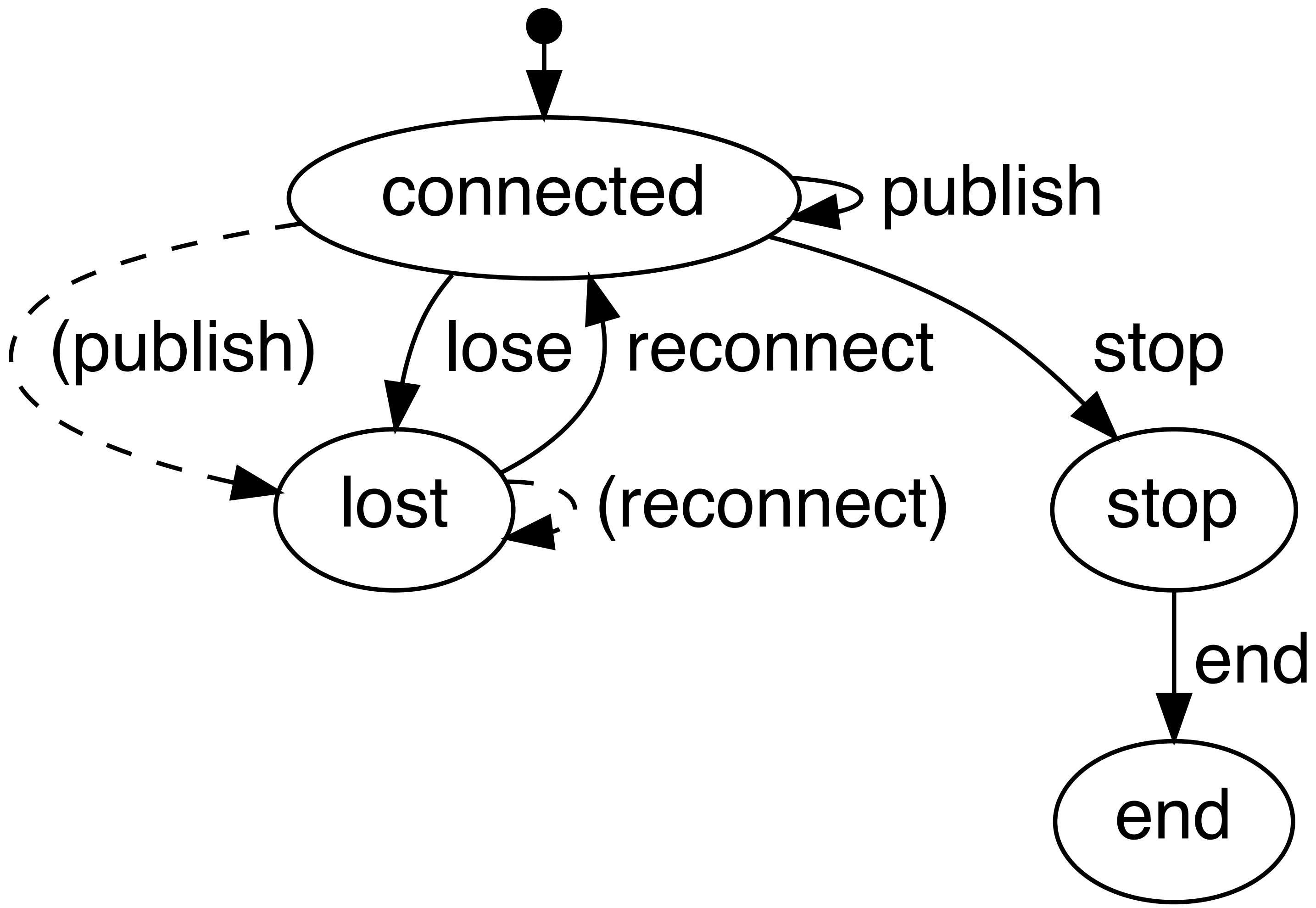


MQTT



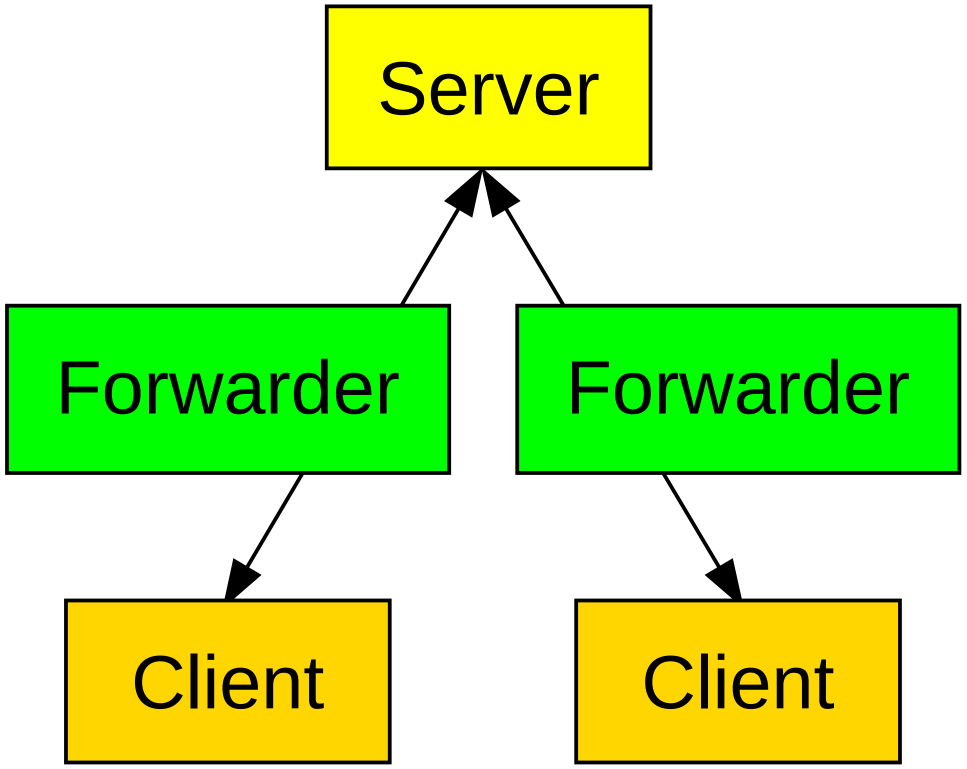
Message Queuing Telemetry Transport:
publish-subscribe protocol used for IoT

MQTT Sender Model: Library usage



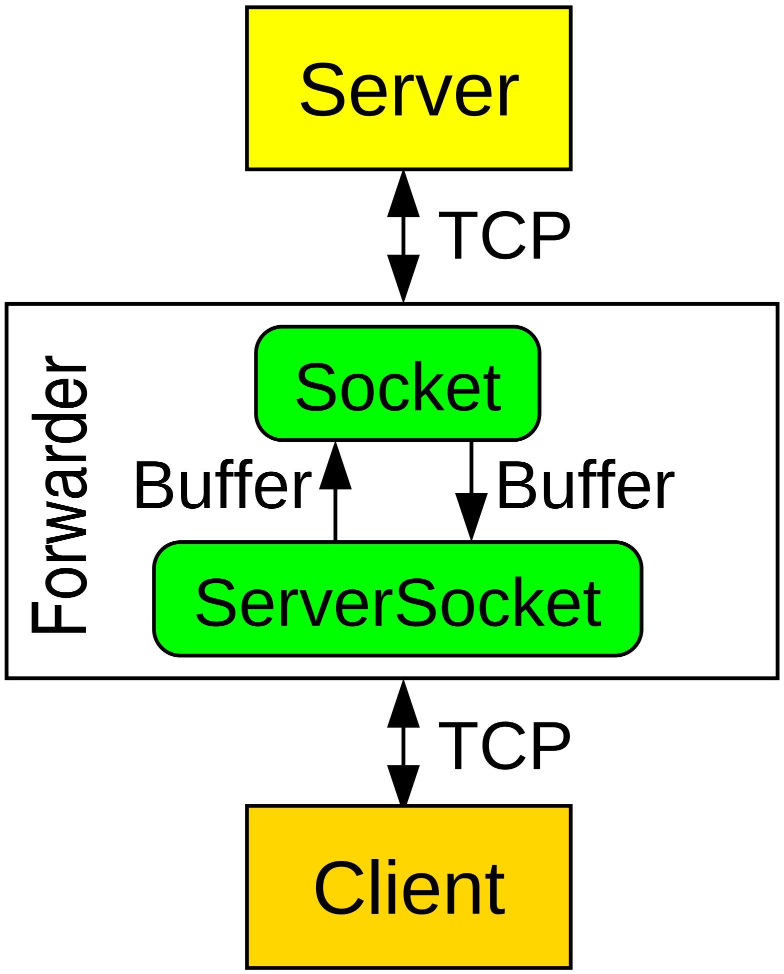
- Transitions are chosen non-deterministically
- Dashed arrows: exceptional outcomes.

Fault Injection



All network connections are captured by packet forwarders.

Packet Forwarders



Model-based simulation:

Modbat runs fault model that sets connection quality parameters.

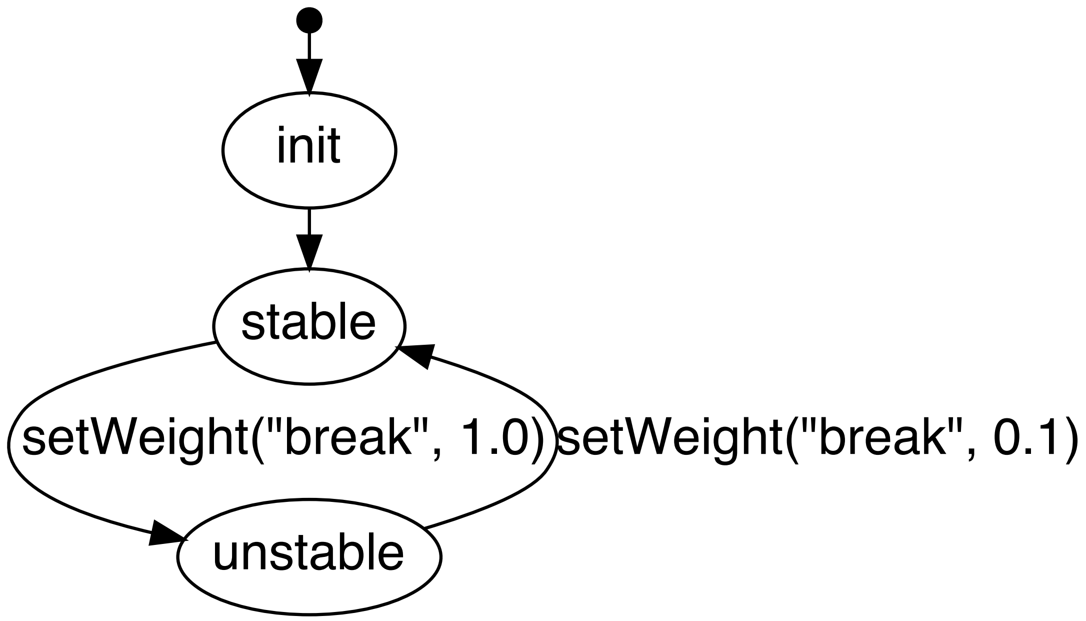
Modbat Extensions for Modelling Fault Injection

stay: stay in current state for given time.

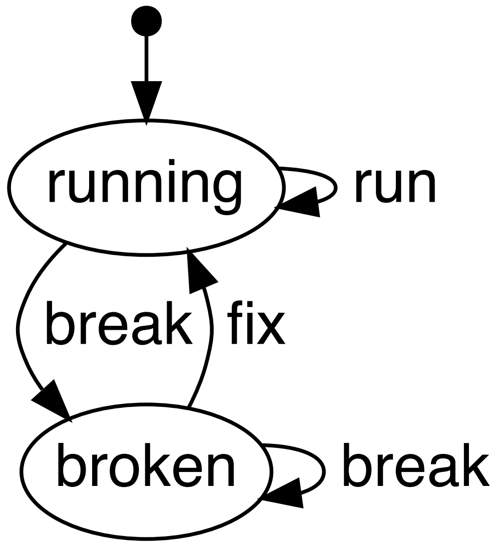
setWeight: adjust transition probability.

invokeTransition: change model state.

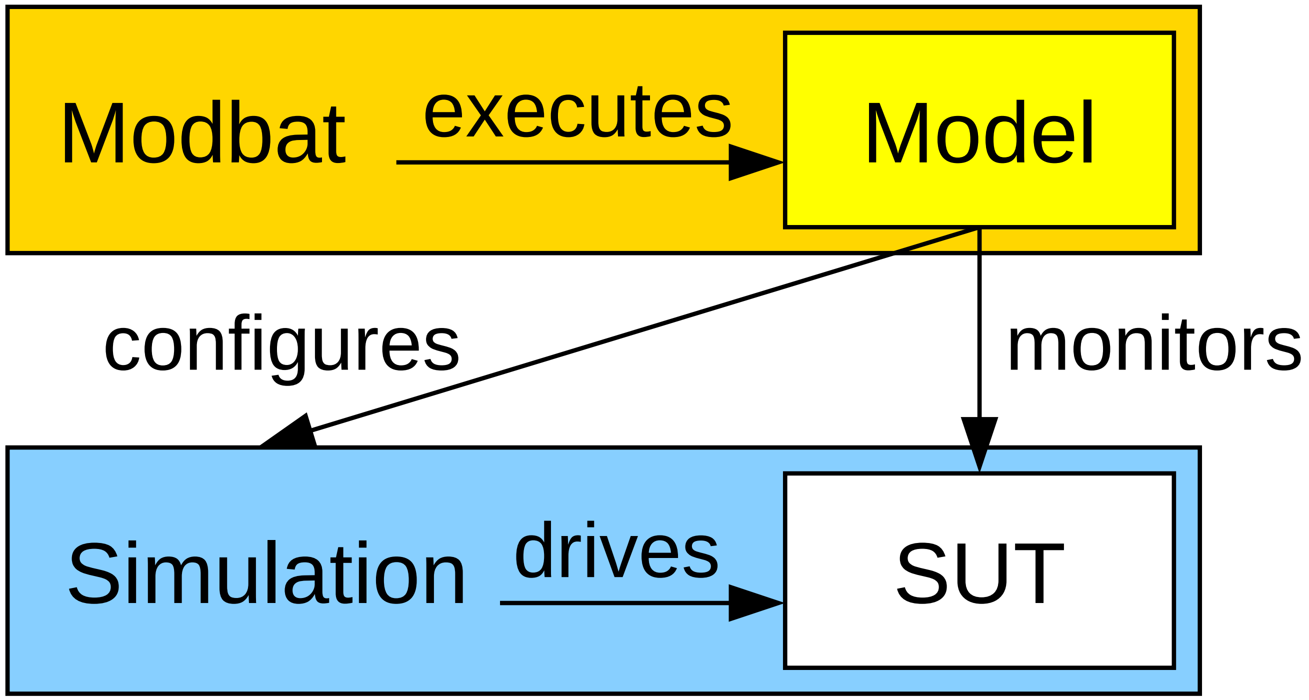
Environment model



Device model



Model-based Simulation

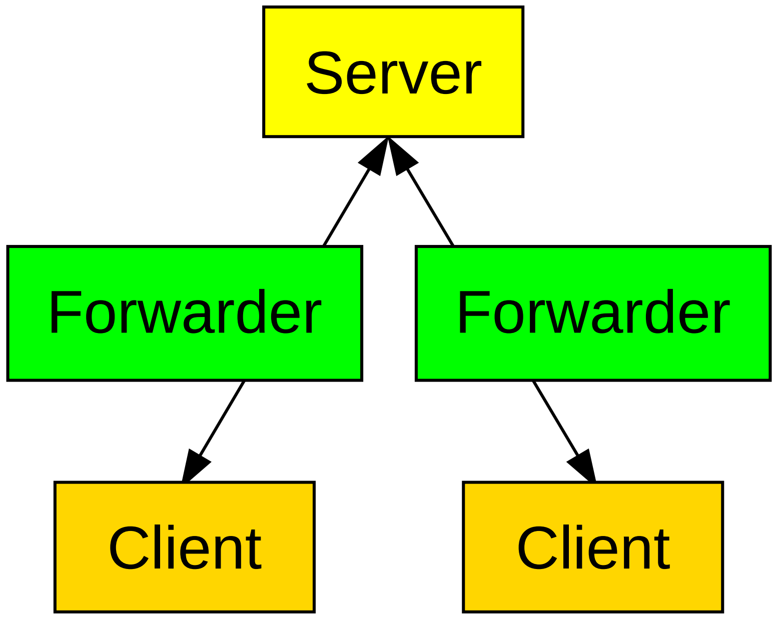


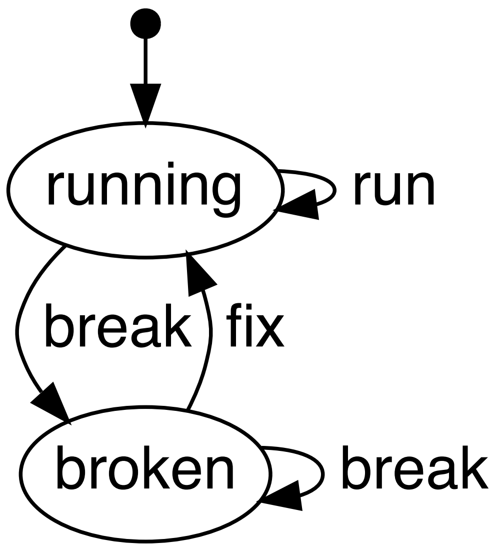
Simulation Results

Message QoS	Message Arrival
0	$published \geq received$
1	$published \leq received$
2	$published = received$

MQTT implementation performs according to specification.

Conclusion





Model-based simulation for MQTT:

- Packet forwarder injects delays and connection loss.
- Model controls simulation settings using tool Modbat.